SYSTEM AND METHOD FOR ROUTER QUEUE AND CONGESTION MANAGEMENT

Tony M. Brewer
5225 Mariners Dr.
Plano, Texas 75093
Citizenship: U.S.A.

Jim Kleiner
13834 Sprucewood Dr.
Dallas, Texas 75240
Citizenship: U.S.A.

Gregory S. Palmer
3012 Mason Dr.
Plano, Texas 75025
Citizenship: U.S.A.

Keith W. Shaw
3229 Dibrell Dr.
Plano, Texas 75023
Citizenship: U.S.A.

RELATED APPLICATIONS

This application is related to co-pending and commonly assigned U.S. Application Serial Number 09/703,057, entitled "System And Method For IP Router With an Optical Core," to co-pending and commonly assigned U.S. Application Serial Number [Attorney Docket Number 59182-P002US-10020639], entitled "System and Method for Router Central Arbitration," to co-pending and commonly assigned U.S. Application Serial Number 09/703,038, entitled "System and Method for Router Data Aggregation and Delivery," to co-pending and commonly assigned U.S. Application Serial Number 09/702,958, entitled "Timing and Synchronization for an IP Router Using an Optical Switch," to co-pending and commonly assigned U.S. Application Serial Number 09/703,027, entitled "Router Network

Protection Using Multiple Facility Interfaces," to co-pending and commonly assigned U.S. Application Serial Number 09/703,043, entitled "Router Line Card Protection Using One-for-N Redundancy" and to co-pending and commonly assigned U.S. Application Serial Number 09/703,064, entitled "Router Switch Fabric Protection Using Forward Error Correction," all filed October 31, 2000, the disclosures of which are incorporated herein by reference.

TECHNICAL FIELD

This application relates to the field of optical communication networks, and particularly to large-scale routers for optical communication networks.

BACKGROUND

A router system has a number of ingress ports that feed information, e.g., data packets to a switching fabric. The switching fabric then routes the information to egress routing ports. In such a system, typically the switching fabric on a per port basis has more bandwidth going into and out of the switch than is actually needed by the ports, such that typically there is more bandwidth capability feeding into an egress port than is feeding out of the egress port. Under these circumstances, queuing within that egress port can become very congested. The queues can fill up, and an intelligent mechanism is needed in order to manage those queues such that traffic is not dropped indiscriminately. In a system having multiple Quality of Service (QOS)levels, particular attention must be paid such that each level has its dedicated queuing space. Yet when the overall queuing space is not heavily utilized, it is desirable that the remaining queues can grow and use more space than they normally would be allotted.

SUMMARY OF THE INVENTION

The present invention is directed to a system and method for managing both instantaneous and time averaged queue congestion in a multi-QOS level queuing structure having a common shared memory pool. Packet payload pointers are stored in multiple queues, each of which represents a distinct QOS priority level, and packet information payloads are stored in the common memory pool. Instantaneously, if an incoming packet causes one of the QOS levels to reach its allocated memory space limitation, then it must be determined whether to discard the packet or not. Algorithms control the drop probability of packets entering the queuing structure.

The instantaneous management algorithm determines the total amount of shared memory space in bytes and monitors the instantaneous actual sizes of the individual queues. The algorithm dynamically calculates a minimum and a maximum queue size for each queue, using the percent utilization of the common memory pool. Any non-utilized common memory space is allocated simultaneously to all of the queues sharing the common memory pool, advantageously providing all queues more than their allocated memory space. A drop probability is calculated from a comparison of actual instantaneous queue size with minimum and maximum queue sizes.

The time averaged congestion management follows a traditional Weighted Random Early Discard (WRED) mechanism. However, both instantaneous and weighted time averaged congestion management are adapted to a multi-level QOS structure as well as to floating point calculations as opposed to a complex multi-dimensional table driven mechanism. In some embodiments the algorithms are implemented in hardware.

Particularly, the present invention is directed to an egress queuing system in a router, such that packets flow out of the queuing structure into a single tributary of a router egress port. The packet flow out of the queuing system is controlled on a QOS priority basis by an egress tributary arbitration algorithm using a rate metering mechanism. Packets are received into the queuing structure from an egress reassembly logic. The queuing structure is replicated for each egress tributary in the router system.

When the queues fill up near their capacity, the present mechanism manages those queues, such that lower priority traffic is dropped more frequently than higher priority traffic, but at the same time lower priority traffic is not completely dropped but rather is only relegated to a smaller percentage of the overall space within that memory system.

Various aspects of the invention are described in co-pending and commonly assigned U.S. Application Serial Number 09/703,057, entitled "System And Method For IP Router With an Optical Core," co-pending and commonly assigned U.S. Application Serial Number [Attorney Docket Number 59182-P002US-10020639], entitled "System and Method for Router Central Arbitration," co-pending and commonly assigned U.S. Application Serial Number 09/703,038, entitled "System and Method for Router Data Aggregation and Delivery," co-pending and commonly assigned U.S. Application Serial Number 09/702,958, entitled "Timing and Synchronization for an IP Router Using an Optical Switch," co-pending and commonly assigned U.S. Application Serial Number 09/703,027, entitled "Router Network Protection Using Multiple Facility Interfaces," co-pending and commonly assigned U.S. Application Serial Number 09/703,043, entitled "Router Line Card Protection Using One-for-N Redundancy" and co-pending and commonly assigned U.S. Application Serial Number 09/703,064, entitled "Router Switch Fabric Protection Using Forward Error Correction," all filed October 31, 2000, the disclosures of which are incorporated herein by reference.

The foregoing has outlined rather broadly the features and technical advantages of the present invention in order that the detailed description of the invention that follows may be better understood. Additional features and advantages of the invention will be described hereinafter which form the subject of the claims of the invention. It should be appreciated by those skilled in the art that the conception and specific embodiment disclosed may be readily utilized as a basis for modifying or designing other structures for carrying out the same purposes of the present invention. It should also be realized by those skilled in the art that such equivalent constructions do not depart from the spirit and scope of the invention as set forth in the appended claims. The novel features which are believed to be characteristic of

the invention, both as to its organization and method of operation, together with further objects and advantages will be better understood from the following description when considered in connection with the accompanying figures. It is to be expressly understood, however, that each of the figures is provided for the purpose of illustration and description only and is not intended as a definition of the limits of the present invention.

BRIEF DESCRIPTION OF THE DRAWING

For a more complete understanding of the present invention, reference is now made to the following descriptions taken in conjunction with the accompanying drawing, in which:

Fig. 1 is a schematic diagram illustrating an egress queuing structure;

Figs. 2A-2D are graphic representations of drop probability curves for the instantaneous management function of the Queue Congestion Management Block of Fig. 1;

Figs. 3A-3D are graphic representations showing how the drop probabilities are modified when the total of the four QOS levels now consume only 80% of the common memory space;

Figs. 4A-4D are graphic representations of calculated drop probability when total common memory space of 50% is utilized; and

Figs. 5A-5D are graphic representations of weighted random early discard probability curves for each of four QOS levels.

DETAILED DESCRIPTION

Fig. 1 is a schematic diagram illustrating an egress queuing structure 10. Packets arrive from egress reassembly logic (not shown) within an egress ASIC through link 101. Prior to the arrival of these packets, a router switching fabric has supplied chunks of information to the egress ASIC, which has separated the individual packets out of the chunks and reassembled the larger packets, such that the packets received through link 101 are reassembled complete packets. Those packets enter Queue Congestion Manager and Weighted Random Early Discard (WRED) Block 12.

Queue Congestion Manager Block 12 determines whether packets should be forwarded to the actual queuing in the egress structure or whether packets should be dropped due to congestion. There are instantaneous congestion issues as well as time averaged congestion issues. Packets emerge out of Queue Congestion Manager Block 12 through four separate links 102-0 through 102-3, each of which carries packets destined for a particular quality of service (QOS) level queue 14-0 through 14-3. Each of queues 14-0 through 14-3 stores individual packet payload pointers; however, the actual packet data for all four queues is stored in a common shared memory pool 13.

An egress tributary arbitration block 16, responsible for determining which packet is next to go out through a link 104, gives selection information to a selection MUX 15, which selects one of the four egress queues 14-0 through 14-3 from which to remove information through links 103-0 through 103-3.

Each egress port can have one or multiple egress tributaries associated with it. For example an OC192c port, which is a 10 gigabit per second (Gb/s) port, can have one tributary feeding that egress port, which would consume all 10 Gb/s of bandwidth. On the other hand, smaller bandwidth tributaries feeding that egress port, for example, four tributaries each having 2.5 Gb/s of bandwidth, would then equal the overall port bandwidth of 10 Gb/s.

In Fig. 1, link 104 represents the output of a single tributary going out of queuing structure 10. In an output port structure, if there are multiple tributaries, then entire queuing

structure 10 would be replicated on each of the tributaries. On link 101 packets would come from a common reassembly logic, and packets designated to each individual tributary would be directed toward the appropriate queuing structure for that tributary. Thus queuing structure 10 would be replicated as many times as the number of tributaries for that particular port. However, the queues from all tributaries share a common memory pool.

A certain bandwidth is pre-allocated to each of the four QOS levels that exist in egress queuing structure 10. An arbitration algorithm controls the flow of data from each QOS level QOS-0 through QOS-3, such that each QOS level receives its fair share of the bandwidth going out of the egress port. However, if there is underutilized bandwidth from any of the QOS levels, which will result in excess bandwidth capacity for that port, then if there are any queues that have packets available, even though these queues have exhausted their current bandwidth allocation, then those queues are allowed to send additional packets out through the egress port.

A rate metering mechanism operates such that periodically, tokens are put into a counter that can for example be 16 or 32 bits in size. Every so many clock periods, a certain number of tokens or count is added to the rate meter. Then, as packets are selected from the queue associated with that rate meter, a certain number of tokens are removed from the rate meter by deducting a quantity associated with the length of the packet, such that the tokens are then associated with a certain amount of bytes out of the packet, or a certain bandwidth through the tributary. Thus, periodically the QOS rate meter is replenished, and then, as packets are extracted from the queue, tokens are deducted from the rate meter.

For example, if a token is equivalent to a byte, then for a 40 byte packet, 40 tokens are deducted from that rate meter. Similarly, the number of tokens added to the rate meter on a periodic basis is equivalent to the amount of bandwidth in bytes per time unit, allocated for that particular QOS level. The rate meter has a maximum number of tokens as well as a minimum number. For example, the minimum number may be 0, in which case if packets are removed, such that the rate meter would go below 0, it may truncate to keep the token level at a value 0. Similarly a rate meter is not allowed to continue to increase its number of tokens

indefinitely if no packets are going out.

In the present implementation an egress tributary arbitration algorithm applies a set of code, namely

(1.0) // find next packet to send, first check token buckets which are positive

```
for (qos=0, pkt=0; !pkt&&qos<MAX_QOS_LEVELS; qos+=1)
if (qosMeter[qos].tokenCnt (trib) >0 & & (pkt=txQueue.unlink_first(trib,qos)))
{qosMeter[qos].decTokenCnt (trib, pkt->posSize())}
```

(1.1)// next check token buckets which are negative

```
for (qos=0; !pkt&&qos < MAX_QOS_LEVELS; qos+=1)
if (qosMeter [qos].tokenCnt(trib)<=0&&(pkt=txQueue.unlink_first(trib,qos)))
{qosMeter[qos].decTokenCnt(trib, pkt->posSize())}
```

Section 1.0 of the arbitration algorithm indexes through each of the four QOS levels and determines if there is a packet in that level and if so, if there are positive rate metering tokens for that QOS level. Thus, if there is a packet available and there are positive QOS tokens available, a packet is extracted from that queue. If there is no packet ready, or if the token meter is negative, then the algorithm proceeds to the next QOS level and repeats the same procedure. This procedure repeats sequentially through all four QOS levels.

Next in Section 1.1 the algorithm cycles through all four QOS levels again to determine whether there are packets ready to go on meters that are negative, i.e., that have already exhausted their quota of bandwidth. Again the algorithm indexes from the highest priority QOS level 0 to lowest priority QOS level 3. The same algorithm could be applied to any number of QOS levels.

Figs. 2A-2D are graphic representations of drop probability curves for the instantaneous management function of Queue Congestion Management Block 12 of Fig. 1. When a packet is ready to be assigned to a queue, these drop probabilities are examined to determine if the packet should be placed in the queue or dropped. Drop probability curves Figs. 2A-2D show drop probabilities calculated when the common memory pool is

completely or very nearly full. Fig. 2A shows that for QOS level 0 queue 14-0 on Curve 201, the drop probability starts out at 0 and stays at 0 until a threshold utilization 202, which occurs at 10% queue utilization. The drop probability then increases to unity at roughly 15% queue utilization and remains there for all higher values of queue utilization. This limits the amount of the common memory space that is available for QOS level 0 to 10% before any packet drops start occurring. The slope between point 202 and 203 is such that a small amount of traffic will be dropped initially, if the amount of space is used in excess of what is allocated to it, but if utilization increases beyond an additional 5%, then all subsequent packets will be dropped.

In Fig. 2B, curve 204 for QOS 1 queue 14-1 starts at a threshold of 20% under an assumption that the QOS level 1 is allotted twice as much bandwidth as is QOS level 0. Similarly QOS levels 2 and 3 are allotted 30% and 40% respectively, as illustrated in Figs. 2C and 2D. Software can configure how much bandwidth each of the QOS levels can consume at the egress port of the router by allocating the number of tokens that are put into the rate meter. The respective bandwidth allocations are reflected in the drop probability curves of Figs. 2A-2D to allocate space on a per QOS level that is consistent with the amount of bandwidth that is able to be consumed by that particular QOS level.

When a queue is full, it is important that each of the QOS levels are allocated their memory space, such that the total of the four memory space utilizations equal 100%. Thus the threshold utilization of 10% on curve 201 of Fig.2A plus the 20% on Fig. 2B, 30% on Fig. 2C, and 40% on Fig. 2D, add to 100%, such that the entire common memory space is utilized fully, but no individual QOS level consumes more than its allocation.

Figs. 3A-3D are graphic representations showing how the drop probabilities are modified when the total of the four QOS levels now consume only 80% of the common memory space. The 20% excess bandwidth can then be used by any QOS level that needs it. In Figs 3A-3D, it is shown that 20% overall excess bandwidth capacity is added to each of QOS levels 0, 1, 2, and 3 illustrated respectively in Figure 3A, 3B, 3C and 3D.

For example in Fig. 3A for QOS level 0, the packet drop threshold 302 occurs at 302

utilization, whereas in Fig. 2A the threshold occurs at 10% utilization, which is a 20% lower utilization than in Fig. 3A. The 20% excess available bandwidth is given simultaneously to each of the four QOS levels, such that QOS level 0, instead of being 10% is now 30%, QOS 1 in Fig. 3B, instead of being 20% is 40%, QOS 2 in Fig. 3C instead of being 30% is 50%, and QOS 3 in Fig. 3D instead of being 40% is now 60%. Thus, any of the four QOS levels could consume 20% more than its allocated memory space, which is acceptable because the common memory pool is only 80% full. As the common memory pool becomes less full, each of the four QOS levels is given a steadily increasing allocation. If at any point in time in Figs. 3A-3D overall utilization were to exceed the 80% full point, then each of these curves would move back to the left, gradually allocating a correspondingly smaller memory space for each individual queue, preventing any single queue from excessively consuming the shared memory space. Thus each of the queues can have its full memory space allocation, but if any queue does not use all of its allocation, then any of the other queues can use that memory space. This is an example of the flexible algorithm described below that actually implements this function in hardware.

Figs. 4A-4D are graphic representations of calculated drop probabilities when the total common memory space is 50% utilized. In Fig. 4A, instead of 10% bandwidth allocated to QOS level 0, 50% is added, resulting in 60% available for QOS level 0. Similarly for QOS levels 1, 2, and 3, 50% is added to each of their respective allocated bandwidths. This implies that overall only 50% of the common memory space is utilized and that in each of these curves the actual individual queue utilization is less than the initial threshold, for example point 402 in Fig. 4A, such that no queue at this point is being overconsumed and such that no packets are being dropped. However, if QOS level 0 actually started to consume 65% queue utilization, some percentage of the packets would be dropped even though overall common memory utilization is only 50%.

A Queue Congestion Management algorithm implemented in hardware in Queue Congestion Manager and WRED block 12 controls instantaneous queue space congestion management for one particular tributary, as follows:

// Physical queue limits

```
(2.1)   memByteCnt = txQueue.totalByteCnt();
(2.2)   queByteCnt = txQueue.queByte.Cnt(pkt->dstTrib(),pkt->qos());
(2.3)   minSize = config.egressMinSize(pkt->qos()) - memByteCnt*
                  config.egressMinSlope(pkt-<qos());
(2.4)   maxSize = config.egressMaxSize(pkt->qos()) - memByteCnt*
                  config.egressMaxSlope(pkt->qos());
(2.5)   if (queByteCnt < minSize)
                  dropProb=0;
(2.6)   else if (queByteCnt < maxSize)
                  dropProb = (queByteCnt - minSize) / (maxSize - minSize);
(2.7)   else dropProb = 1;
```

The hardware monitors the instantaneous actual sizes of the queue utilizations to make these calculations. The total amount of memory space in bytes is stored in a hardware register *memByteCnt*, as shown in Equation 2.1. Every time a packet is assigned to any of the queues or extracted from a queue, a queue byte count hardware register *queByteCnt* is updated to reflect the total amount of bytes in each of the four individual QOS queues for that tributary, as shown in Equation 2.2. The threshold intercept point, for example point 202 in Fig. 2A, is represented by a configuration value *config.egressMinSize*, which is stored in a hardware CSR and is indexed by QOS level. Point 202 is the allocated minimum threshold of space allocation for QOS level 0. Regardless of whether the queue is full, that amount of space is always available to that particular QOS level. However, as the total common memory utilization is decreased, point 202 moves to the right, as illustrated in Figs. 3A and 4A, to allocate more space for an individual queue.

In Equation 2.3 the minimum queue size *MinSize* is calculated by representing the slope in curve 201 by another configuration value *config.egressMinSlope*, which is then multiplied by the memory byte count *memByteCnt* to modify the *config.egressMinSize* value.

Equation 2.4 calculates the maximum size *maxSize* starting with a configured value *config.egressMaxSize*, which is the value to be used when the queue is fully consumed and is indexed by the appropriate QOS level. That value is then modified by *config.egressMaxSlope* multiplied by the memory byte count *memByteCnt* to displace that value to the right as the common memory utilization decreases.

Next a comparison of the actual queue size for that queue is performed to compute the drop probability. In Equation 2.5 if the actual queue size is less than the calculated minimum size, then the drop probability is set to zero. If the actual queue size is between the maximum and minimum size according for example to curve 201, then the drop probability is calculated by using the slope of the curve, namely *queByteCnt* less *minSize* divided by the quantity *maxSize* minus *minSize*, as shown in Equation 2.6. This calculates a drop probability along curve 201 for example from point 202 to point 203. Finally in Equation 2.7, if the queue size is greater than the value at point 203, then the drop probability is set to unity. Queue Congestion Manager and WRED block 12 of Fig. 1 calculates this drop probability and also calculates a random number between the value of 0 and 1.0, to determine if the individual packet should be dropped or passed on through and then queued in the appropriate QOS level queue.

The quantities that are used to calculate the drop probability can extend over a wide dynamic range, including fractional numbers as well as very large numbers, depending upon how they are used in these equations and the actual size of the queues and packets. Instead of using integer values typically used in hardware design for these types of equations, a small floating point format is actually used. This floating point format uses a four-bit normalized mantissa, which implies that the most significant bit of the mantissa always has the value 1. The four-bit mantissa is normalized such that the value always is between the values of 0.5 and 1.0. The floating point format has a 6-bit exponent biased such that it is between 0 and a maximum value of 63. This allows the exponent to have a range of -32 to +31, obtained from taking the 6-bit value and subtracting off a bias value of 32. In the equations, only the positive numbers are required, such that no sign is needed for the floating point numbers. All

of the algorithms require these numbers greater than or equal to zero. With such a small mantissa, multiply and divide operations are accomplished with simple look up tables, for example two 3-bit inputs result in one 4-bit output. Since the most significant bit of the mantissa always has a value of one, it does not need to be explicitly represented, but is rather implied, and does not need to be part of the two 3-bit inputs in table lookups.

Figs. 5A-5D are graphic representations of weighted random early discard (WRED) probability curves for each of four QOS levels QOS-0 through QOS-3. In Fig. 5A, assuming a 10% overall bandwidth allocated to QOS level 0 as in Figs. 2A-2D, curve 501 has a threshold 502 which is less than 10% utilization. Threshold 502 is the value of the average common memory space that is being used by QOS level 0 at which a small number of packets begin to be dropped. At a point 503 the amount of memory space consumed by QOS level 0 has reached a maximum, such that the drop probability at point 503 immediately goes vertically to unity, as indicated by point 504. These curves are implemented by the WRED algorithm, which is well known in the literature. In the present embodiment, however, the WRED algorithm is applied concurrently to four different QOS levels all sharing a common memory space, such that a certain portion of the overall memory space is allocated to each of the QOS levels. For example, in Figs. 5A-5D the maximum memory space allocated to QOS level 0 is 10%, whereas QOS levels 1, 2, and 3 receive respectively 20%, 30%, and 40%. These respective allocations of shared memory pool space are the same as the corresponding queue bandwidth allocations of each QOS level queue, as described above in connection with Figs. 2A-4D.

In the present embodiment, the WRED algorithm applied to egress queuing is:

// WRED for egress queueing

```
(3.1)   avgByteCnt = txQueue.avgByteCnt(pkt->dstTrib());
(3.2)   prob1 = avgByteCnt * config.egressWredConst1(pkt->qos()) -
                    config.egressWredConst2(pkt->qos());
(3.3)   dropProb = pkt->size() * prob1;
(3.4)   wredCnt = txQueue.wredCnt(pkt->dstTrib(), pkt->qos());
(3.5)   if (!pkt->wredDisable () & &
                    (avgByteCnt > config.egressQueMaxThresh(pkt->qos()) | |
                    avgByteCnt > config.egressQueMinThresh (pkt->qos()) & &
                    rndDrop - rndDrop * wredCnt * dropProb <dropProb() {

// drop packet

(3.6)   txQueue.wredCnt (pkt->dstTrib(), pkt->qos(), 0);
        deletepkt;
        } else {

        txQueue.append(pkt);
(3.7)   txQueue.wredCnt(pkt->dstTrib(), pkt->qos(), wredCnt+1);
        }
    }
```

In Equation 3.1 of the WRED algorithm, an average byte count *avgByteCnt* is obtained for the particular queue into which a packet is being placed, that is indexed by both the destination tributary and the QOS level of the incoming packet. The average byte count is calculated by determining the time weighted average for the actual byte count for that particular queue. In Equation 3.2 a probability *prob1* is calculated by multiplying the average byte count *avgByteCnt* by a constant *config.egressWredConst1* and then subtracting a different constant *config.egressWredConst2*. These constants are a base and a slope corresponding to points 502 and 503 in figure 5A. In Equation 3.3 the drop probability *dropProb* is calculated by multiplying *prob1* by the size of the packet, such that the smaller the packet the less likely it will be dropped. This is desirable to minimize the likelihood that a TCP/IP acknowledge packet will be dropped. In equation 3.4, the *wredCnt* value for the

target queue is obtained. The *wredCnt* is used to make the interval between dropped packets uniform in delay. The *wredCnt* is incremented by one each time a packet is not dropped, and set to zero when a packet is dropped. The conditional expression of equation 3.5 determines if a packet is a candidate to be dropped by examining the WRED disable bit associated with the packet, and be determining if the average byte count, *avgByteCnt*, is within the region where packets are dropped. Average byte counts less than *egressQueueMinThresh* are not dropped. Average byte counts in the range from *egressQueueMinThresh* and *egressQueueMaxThresh* are dropped using the specified probability function.

Equation 3.6 is performed when a packet is to be dropped. The equation sets the *wredCnt* for the specific target queue to the value zero. Likewise, equation 3.7 is performed when a packet is appended to a queue. Equation 3.7 is used to increment the value of *wredCnt* by one. Once a packet is dropped, it is deleted entirely from the router. The entire algorithm is implemented in hardware using the floating point format as described above. Additionally, Queue Congestion Manager Block 12 checks common shared memory space 13, and if shared memory space has been fully consumed, then drops the packet independently of the instantaneous congestion management as well as random early discard management.

Although the present invention and its advantages have been described in detail, it should be understood that various changes, substitutions and alterations can be made herein without departing from the spirit and scope of the invention as defined by the appended claims. Moreover, the scope of the present application is not intended to be limited to the particular embodiments of the process, machine, manufacture, composition of matter, means, methods and steps described in the specification. As one of ordinary skill in the art will readily appreciate from the disclosure of the present invention, processes, machines, manufacture, compositions of matter, means, methods, or steps, presently existing or later to be developed that perform substantially the same function or achieve substantially the same result as the corresponding embodiments described herein may be utilized according to the present invention. Accordingly, the appended claims are intended to include within their

scope such processes, machines, manufacture, compositions of matter, means, methods, or steps.